Secure Android

DASSD, Sept 2022 | C-DAC, Hyderabad

TABLE OF CONTENTS

Let's Begin!

01
Activity

Android Activities

- In Android, an **Activity** represents a single screen with a user interface (UI) and it will acts an entry point for the user’s to interact with app.

- For example, a contacts app that is having multiple activities like showing a list of contacts, add a new contact, and another activity to search for the contacts. All these activities in the contact app are independent of each other but will work together to provide a better user experience.

Android Activities

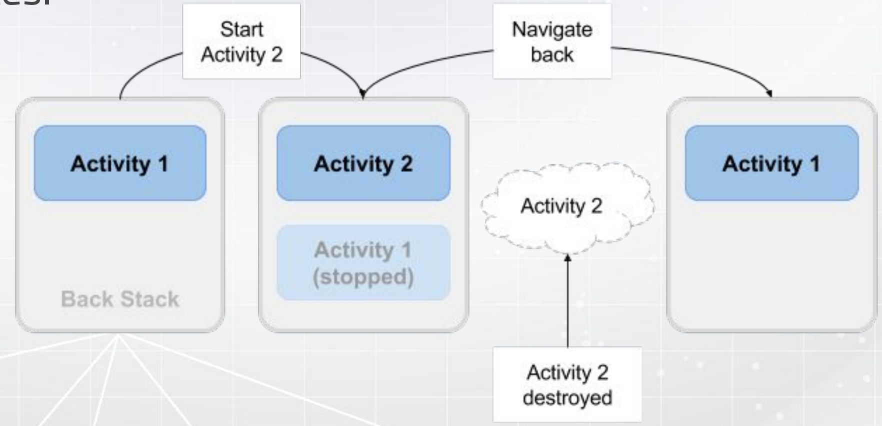
- The component that most Android developers learn about first is the Activity.

- An Activity is analogous to a single screen displayed on the device to a user that is composed of what the user actually sees. When building an app, you create classes that derive from the Activity base class and typically contain one or more Views, which are objects that users can interact with in some way, such aa reading text displayed within the View, clicking a button, and so on.

- In proper application development terminology, the set of Activities is your application’s presentation layer.

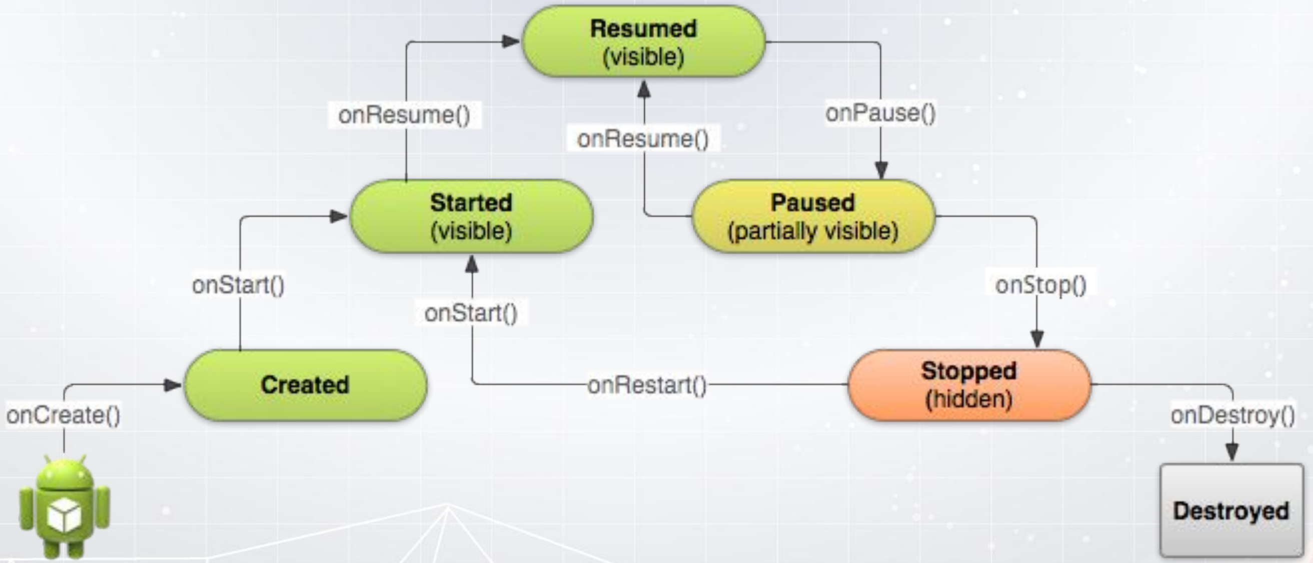
Activity Lifecycle

- The activity lifecycle is the set of states an activity can be in during its entire lifetime, from the time it's created to when it's destroyed and the system reclaims its resources.
- As the user interacts with your app and other apps on the device, activities move into different states.



Activity Lifecycle

- When an Activity transitions into and out of the different lifecycle states as it runs, the Android system calls several lifecycle callback methods at each stage.



Activity Lifecycle

- When an Activity transitions into and out of the different lifecycle states as it runs, the Android system calls several lifecycle callback methods at each stage.

- All of the callback methods are hooks that we can override in each of our Activity classes to define how that Activity behaves when the user leaves and re-enters the Activity.

- (You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.)

Activity Lifecycle

- The Activity class defines the following callbacks i.e. events.

Callback	Description
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

How to Create

- In Android, an **Activity** represents a single screen with a user interface (UI) and it will acts an entry point for the user's to interact with app.

```
public class MainActivity extends Activity {
  // rest of the code
}
```

Points About Activities

- An activity class loads all the UI component using the XML file available in **res/layout** folder of the project.
- Following statement loads UI components from res/layout/activity_main.xml file:

```
setContentView(R.layout.activity_main);
```

Points About Activities

- An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your AndroidManifest.xml file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the **MAIN** action and **LAUNCHER** category as follows:

```
<activity
 android:name=".MainActivity"
 android:label="@string/title_activity_main" >
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER"/>
 </intent-filter>
```

02
Service

Android Services

- In Android, a **Service** is a component that keeps an app running in the background to perform long-running operations based on our requirements.

- For Service, we don't have any user interface and it will run the apps in background like play music in background when the user in different app.

```
public class ServiceName extends Service {
  //rest of the code
}
```

Android Services

- A Service is an Android component that is designed for background processing. These components can run when your app is not visible (that is, none of its Activities are being displayed to the user and are not active). Services typically either carry out some computations or update the data sources for your app.

- For example, if you were to build an email client app, a Service would run in the background to open up connections to the configured mail servers every so often, check for new mail, download it, and store new messages in the local database. Services can also be used to update Activities directly, to provide updates to the user directly via a Notification, or really for any other type of task that needs to occur in the background.

Android Services

- In Android, a **Service** is a component that keeps an app running in the background to perform long-running operations based on our requirements.

- For Service, we don't have any user interface and it will run the apps in background like play music in background when the user in different app.

```
public class ServiceName extends Service {
  //rest of the code
}
```

03
Broadcast Receiver

Android Broadcast Receivers

- In Android, Broadcast Receiver is a component that will allow a system to deliver events to the app like sending a low battery message to the app. The apps can also initiate broadcasts to let other apps know that required data available in a device to use it.

- Generally, we use Intents* to deliver broadcast events to other apps and Broadcast Receivers use status bar notifications to let the user know that broadcast event occurs.

```
public class MyReceiver  extends  BroadcastReceiver {
   @override
   public void onReceive(context,intent){}
}
```

Android Broadcast Receivers

- A Broadcast Receiver is a type of component that listens for system messages called Intents. An Intent can be thought of as a request for a certain action to take place. Apps can create Intents and either send them directly to a specific component (usually an Activity or a Service) or broadcast them system-wide to all apps that are running.

- A Broadcast Receiver is a component that can receive these systemwide broadcasts and act upon them; it can choose to listen for all broadcast Intents or set up filters so that it receives only Intents for the specific actions it cares about (and would, presumably, take action upon). As with most broadcast systems, more than one Broadcast Receiver can receive, and act upon, a single Intent.

04 Content Provider

Android Content Providers

- In Android, Content Providers are useful to exchange the data between the apps based on the requests. The Content Providers can share the app data that stores in the file system, SQLite database, on the web or any other storage location that our app can access.

- By using Content Providers, other apps can query or modify the data of our app based on the permissions provided by content provider. For example, Android provides a Content Provider (ContactsContract.Data) to manage contacts information, by using proper permissions any app can query the content provider to perform read and write operations on contacts information.

Android Content Providers

- A Content Provider is a component designed to share data across apps. You can think of a Content Provider as the public interface to your databases, allowing other apps to connect and either run queries (retrieve data) or issue updates (store data). A Content Provider typically is used as a front end for a database created using the Android standard SQLite database system. As Content Providers are typically used to share data across apps, properly securing them so that appropriate apps can access specific data is **critical**.

```
public class contentProviderName extends  ContentProvider {
   @override
   public void onCreate(){}
}
```

Android Intents

- Intents are the primary way for apps to send messages to other apps (and can also be used to send messages to other components within the same app). To send a message, an app creates an Intent object that signifies that it wants a certain action to take place. This Intent object can specify a specific Activity or Service that you want to start up, or it can specify a specific piece of data. In the later case, the Intent can either specify which components should perform an action on that piece of data, or just ask that someone should. This means that Android allows for Intents that have specific recipients (the Activity or Service that it wishes to interact with) as well as Intents that are broadcast throughout the system to any components that may be listening.